From: *Proceedings of the Third IEEE International Workshop on Systems Management*, Newport, RI, USA, April 1998.

A Management Interface for Distributed Fault Tolerance CORBA Services[*]

Jürgen Schönwälder,[†] Sachin Garg,[‡] Yennun Huang,[‡]
Aad P. A. van Moorsel[‡] and Shalini Yajnik[‡]

[†] TU Braunschweig
Bültenweg 74/75
38104 Braunschweig, Germany
`schoenw@ibr.cs.tu-bs.de`
`http://www.ibr.cs.tu-bs.de/~schoenw`

[‡] Bell Laboratories Research
Lucent Technologies
600 Mountain Ave., Murray Hill, NJ 07974, USA
`sgarg,yen,aad,shalini@bell-labs.com`
`http://www.bell-labs.com/~sgarg,~yen,~aad,~shalini`

ABSTRACT

CORBA is becoming an increasingly important middleware platform for distributed software applications in areas such as telecommunications. The DOORS fault-tolerance service is a CORBA service that adds fault tolerance to CORBA applications. In this paper we discuss the benefits of adding a management interface to services like DOORS. We design this interface and provide an implementation based on SNMP. The management interface collects and displays data about DOORS, and feeds back information about the underlying computing platform to DOORS, to improve its decision making. In the design, we clearly delineate DOORS and the management components, and provide an extended agent to allow communication between CORBA and SNMP. We will discuss the interaction between CORBA and SNMP in detail, and will provide a MIB specifying an information base for DOORS.

[*]This work was carried out when Jürgen Schönwälder was consultant at Bell Labs Research in Murray Hill, summer 1997.

I Introduction

As applications are moving towards more object-oriented distributed platforms, distributed object middleware technologies like the Common Object Request Broker Architecture (CORBA) [8] are becoming increasingly popular. Several projects in financial, telecommunications and health industry currently take advantage of the distributed object-oriented programming environment offered by the CORBA standards.

Although the CORBA middleware eases the development of distributed applications, the current version of the CORBA standards do not address the reliability and availability requirements found in many applications, especially in the telecommunications world. In order to improve the reliability and availability of applications, some researchers have implemented Object Request Brokers (ORBs) based on the concept of group communication and virtual synchrony [5, 6]. At Bell Laboratories we have taken a different, service-based, approach by extending the existing set of CORBA services with a fault tolerance service, called Distributed Object-Oriented

Reliable Service (DOORS) [1]. The DOORS system is implemented as a collection of interacting CORBA objects, that detect CORBA object failures and host failures, and recover CORBA objects gracefully from such failures. An application developer who wants to improve the reliability of an application uses the DOORS service to implement fault-tolerant CORBA objects.

In this paper we introduce DoorMan, a management interface to DOORS. The primary task of DoorMan is to monitor DOORS as well as its underlying computing system. Monitoring DOORS is important to visualize, understand and fine-tune the functioning of DOORS, while monitoring of the underlying system allows DOORS to tolerate failures using the failure-prevention approach, e.g., it can take corrective action and migrate CORBA objects if it detects that an object's local host is suspected to crash soon [3].

In the design of DoorMan we carefully delineate DOORS and DoorMan, each having its own functionality and software modules. DoorMan does not change or take over any of the functions of DOORS, nor does it assume any responsibility about decisions related to the fault-tolerance mechanisms. As a consequence, DOORS performs its intended functions even in the absence of DoorMan. Furthermore, the delineation allows to implement DoorMan using off-the-shelf management components, which come with the added advantages of open and standardized technology. Our implementation is based on the Simple Network Management Protocol (SNMP) [12, 14].

The main focus in the design of DoorMan lies in the interfacing between the CORBA world of DOORS and the SNMP world of DoorMan. To that end, we develop a gateway formed by an extended agent, that is able to bind with the 'ReplicaManager.' (The ReplicaManager is one of the DOORS objects, as explained in Section II.B.) In our design, adaptations to DOORS to facilitate the management interface are restricted to the ReplicaManager object. To specify the information that DoorMan can request from DOORS, we define a DOORS management information base (the DOORS MIB). Vice versa, for the information DoorMan feeds back to DOORS, we discuss the appropriate abstraction level to bridge the gap between the detailed view of the manager and the focused view of DOORS.

The proposed management interface may have applications beyond DOORS. In general, adding separate management utilities to CORBAServices seems a natural, flexible and potent approach to improve the operation of the service. Doing this results in a three-step architecture: the base CORBA ORB, a CORBAService, and the service management. This paper will try to illustrate the appropriateness of such a design for the DOORS reliability service. It should be noted that the approach we follow is fundamentally different from the integration between CORBA and SNMP in the CORBA-SNMP gateway [7] and that proposed in the Joint Inter-domain Management working group, a joint activity of The Open Group and the Network Management Forum [10]. There, a mapping between CORBA and SNMP is defined that allows developers to implement SNMP agents and management applications in CORBA. In doing so, SNMP management applications can be implemented using CORBA without much SNMP knowledge. In our approach, however, a CORBAService can be managed by an SNMP management application with minor knowledge about CORBA.

This document looks at the management aspects of the Distributed Object-Oriented Reliable Service for CORBA. Section II gives a short overview of CORBA and the functionality and the components that make up DOORS. Section III describes how DOORS can be extended so that management systems can monitor its operation, as well as how the DOORS system can take advantage of the information available from a management system. Section IV discusses implementation details and Section V states conclusions and lists some topics for future work.

II CORBA and DOORS

In this section, we will give some basic concepts behind CORBA and then briefly describe the design and functionality of DOORS.

A CORBA

CORBA is an emerging object middleware technology which supports the development of distributed object-oriented applications. It is an architectural standard proposed by a consortium of industries called the Object Management Group (OMG) and its operational model is based on RPC-style client-server communication. The core of the CORBA architecture is the Object Request Broker (ORB). The ORB acts as the object bus which locates the server for a client request and maintains the communication between the server and the client. The ORB provides server activation and server location transparency to the client. Various vendors have implemented the CORBA standards and there are a number of CORBA products, e.g. IONA's Orbix [4], Visigenic's VisiBroker, in the market today. In this paper, we will not go into the details of the CORBA architecture. Interested readers are referred to the vast literature present in the area [4, 8, 9].

The CORBA standards also propose additional services, called CORBAServices [9], built and supported on top of the ORB. These services, like Naming Service, Trader Service, LifeCycle service, provide some useful functionality which can be used by all types of CORBA applications. DOORS (Distributed Object-Oriented Reliable Service) [1], developed at Bell Laboratories, is one such CORBA service.

B DOORS

DOORS provides a means for an application developer to enhance the availability and reliability of his/her application built on top of CORBA. DOORS adds embedded fault-tolerance support to CORBA objects in order to tolerate host crashes, object crashes and object hangs [1]. It uses replication of objects as the mechanism to support high availability. DOORS provides failure detection, failure recovery, and replica management at the object level. An application developer has the flexibility to choose the degree of reliability and availability by selecting from an array of choices for replication strategies, degree of replication, detection mechanisms, and recovery strategies that are best suited to the application object at hand. The DOORS interface provides methods through which application objects can register their reliability requirements and then DOORS takes actions based on the options specified during registration.

There are three main modules in DOORS, which work together to provide the functionality described above - WatchDog, SuperWatchDog and the ReplicaManager. We discuss in brief their individual functionality and their interactions with each other. The modules are shown in Figure 1.

- The **WatchDog (WD)** module runs on each host in the network and detects object crashes and hangs for object servers running on the local host. The WatchDog can use polling to detect object crashes and heartbeats to detect object hangs. The Watchdog also performs local recovery actions.

- The **SuperWatchDog (SWD)** module is centralized and is responsible for detection of host crashes and hangs. It receives heartbeats at regular intervals from all the WatchDogs in a network domain. Host crashes are detected if heartbeats from a previously registered WatchDog do not arrive in a given time interval.

- The **ReplicaManager (RM)** module is also centralized and as the name suggests, it is responsible for management of object replicas. An object can register with the ReplicaManager with a requested degree of replication, a given replication style, and a list of possible locations for the replicas. The ReplicaManager manages the initial placement and activation of these object replicas and also controls the migration of the replicas during object failures. It keeps track of how many replicas of an object exist in a network domain, on which hosts they are running, the status of each replica, and the number of failures seen by the replica on a given host. This information is maintained in a table at the ReplicaManager.

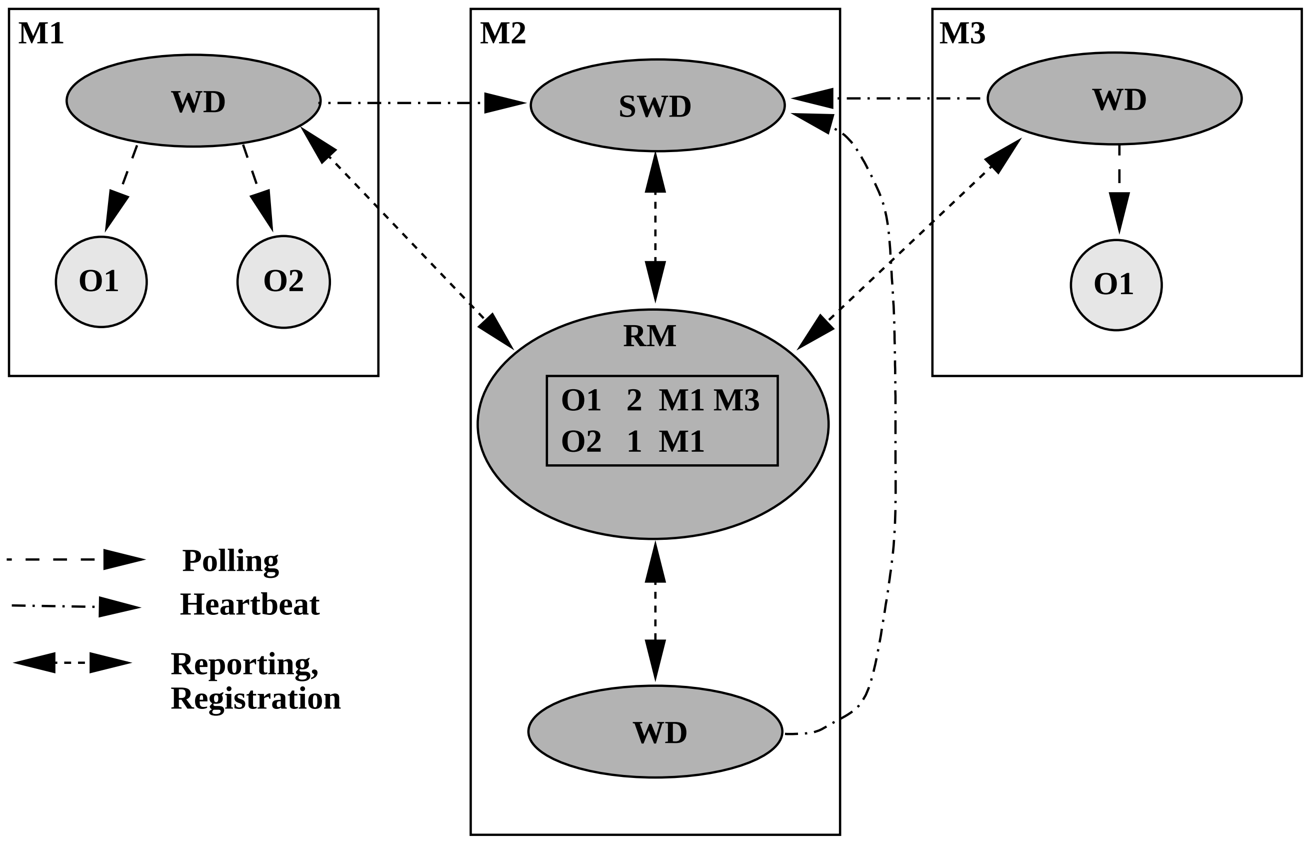


Figure 1: Structure of DOORS

Figure 1 shows how the the modules are used to control two application objects. Object O1 on host M1 registers with the ReplicaManager with replication degree two and with possible hosts M1 and M3. RM registers O1 with WDs on M1 and M3. The WD on M3 detects that there is no running copy of O1 on the local host and starts a copy on M3. After activation, the WD watches over the replica through polling. Similarly, a single replica of the object O2 is run on M1 and all the replicas are watched over by their local WDs through polling. As shown in the figure, the WatchDogs on each host send heartbeats to the central SuperWatchDog.

The ReplicaManager module maintains all the state of the DOORS system. The table stored in the ReplicaManager has the information about all the objects and their replicas in the system, their registration options, their location, their status, the number of times they have failed on a particular host, etc. The information in this table will be monitored by the management system that interfaces with DOORS. To prevent loss of this state due to failure of the ReplicaManager, this table is periodically checkpointed.

III DoorMan Design

In the design of the management interface for DOORS we clearly delineate functionality that belongs with DOORS and functionality that belongs with management. DoorMan operates in such a way that DOORS does not depend in any way on the presence of the management system. That is, DOORS is able to function properly even if no manager is available although its efficiency in improving the reliability of distributed objects may be reduced. A clear delineation of the management components also allows us to fully deploy existing management technology, since this technology will not interfere with DOORS. (We benefit from this in our implementation by adopting SNMP.) The main challenge in the design of DoorMan then is how it interfaces with DOORS, a topic we will now discuss in detail.

A Interfacing between DOORS and DoorMan

DoorMan interacts with DOORS bi-directionally. First, DoorMan obtains data about DOORS' operation, such as the names, number and type of registered objects, and the location and status of the WatchDogs and SuperWatchdogs. The obtained data can be used to analyze DOORS, as well as to display the status of DOORS in a graphical interface. Secondly, DoorMan collects additional information about the underlying computing platform and feeds it back to DOORS. The data collected can, for instance, correspond to the status of the OS resources or the communication links.

DoorMan interfaces with DOORS *only* through the ReplicaManager. We think this is a natural decision, because, as we have seen in Section II.B, the ReplicaManager contains a table with all data DoorMan might want to collect about the DOORS fault-tolerance mechanisms. In addition, in this way we localize the changes in DOORS to a single module, the ReplicaManager. To establish the information exchange from the ReplicaManager to DoorMan, we add a single method to the ReplicaManager's IDL. The method, when called, delivers all information in the ReplicaManager to DoorMan. In Section IV we discuss this in more detail when introducing the SNMP implementation of DoorMan.

B Passing Information from DoorMan to DOORS

The other way around, DoorMan may want to pass information back to DOORS. Again, information is allowed to be passed only to the ReplicaManager, not to any other component of DOORS. Inevitably, we have to add methods to the ReplicaManager IDL to be able to receive information from DoorMan, and to act upon it appropriately.

The ReplicaManager has a view which only includes the components that are involved in DOORS, while the manager has a much more detailed view about underlying resources and the communication infrastructure. The consequence of passing information from DoorMan to DOORS is that the view of DOORS must be enhanced and the management information sent from the manager to DOORS should use an abstraction level that matches the decision algorithm used by the ReplicaManager. We therefore introduce a 'health' status of resources, and the DoorMan manager abstracts the health status from the collected system data, and passes it to the ReplicaManager. (When DoorMan is not present, 'health' corresponds to 'up' or 'down' of objects and hosts.) The ReplicaManager then determines what to do with the information.

DoorMan derives the higher-level health information out of the very detailed information sources. In related research we have identified which objects are statistically relevant for the 'health' of resources [3]. Based on such results, one can select a suitable information base for DoorMan in order to improve the decision making in DOORS.

IV SNMP-Based DoorMan Implementation

Based on the design choices presented in Section III, we now discuss the SNMP-based implementation of DoorMan. Since the DoorMan management component is to a large extent independent from DOORS, we are able to use standard technology to implement DoorMan. Choosing SNMP allows us to use existing agent implementations, the sub-agent development kit available for Solaris machines, as well as the Tnm Tcl extension [13]. The DoorMan implementation is for SUN Solaris, and DOORS is implemented using Orbix (IONA's implementation of CORBA) for Solaris [4].

As dictated by SNMP, we develop an agent and a management component. Most interesting is the agent architecture, which runs on the same host as the ReplicaManager. In its implementation one must consider how to establish interaction between SNMP and CORBA, and what MIB [11] to define for exporting a DOORS-dependent information base.

A Agent Architecture

Figure 2 shows the agent architecture for DoorMan. It consists of an SNMP agent, and a DOORS subagent. The SNMP agent is the one provided by SUN for Solaris systems, and for the subagent implementation (in C) we used the Solaris subagent development kit. The subagent operates as a CORBA client, and uses a sub-agent protocol like AgentX [2] to present the management information as part of the SNMP management information base exported by the local system. Being a CORBA client means that the subagent is compiled with the CORBA client library and the ReplicaManager's stub, and is able to obtain a reference to the ReplicaManager. The introduction of a subagent simplifies the implementation of management applications, since it provides transparent access to a DOORS MIB implementation. In addition, subagents simplify the security administration of the network management subsystem considerably.

The IDL interface of the ReplicaManager has been extended to provide a method call to extract information about registered client objects and existing replicas. All this information is retrieved using a single method call, in order to keep the number of interactions low, and the changes to the ReplicaManager limited. To further limit the overhead, the sub-agent caches the retrieved data so that management requests can be satisfied from the cache if they come in sequence. This also helps to enhance the consistency of the data as seen by the manager.

The ReplicaManager IDL must also allow the subagent to feed back information to DOORS, and the ReplicaManager must use this information to enhance its decision making. The DoorMan manager processes the data collected, eventually converting to numbers which represent the health of the various resources. It requires experimental investigation to determine the most suitable way of converting the detailed data in health indicators for the system components.

B DOORS MIB

The management information exported to DoorMan is defined in the DOORS MIB (Figure 3). The DOORS MIB contains a specification of all the information the ReplicaManager has about the functioning of DOORS. In Figure

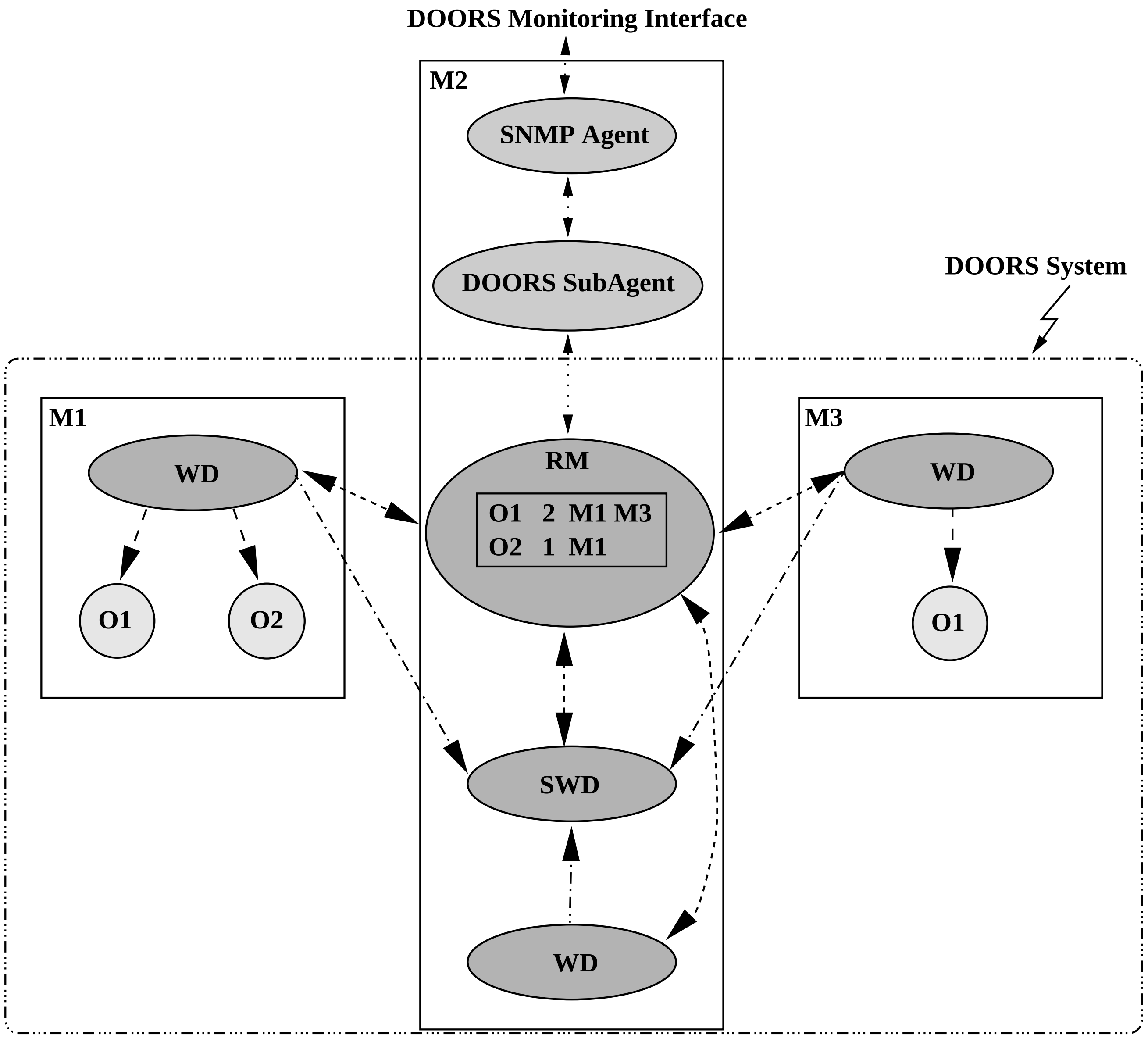


Figure 2: SNMP agent and DOORS subagent, inside the host running the ReplicaManager.

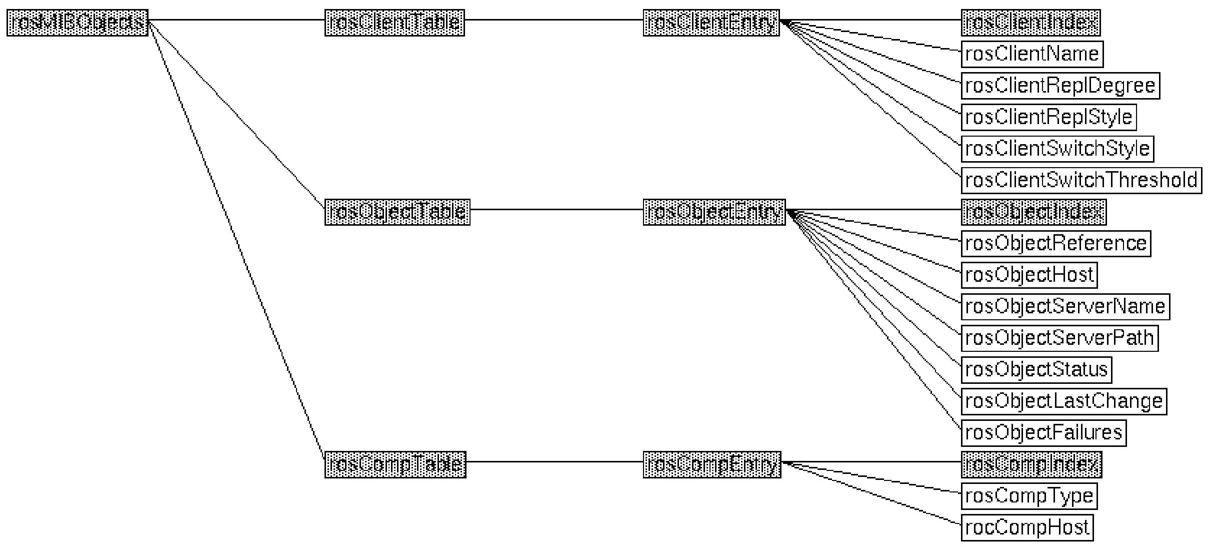


Figure 3: Structure of the DOORS MIB.

3, the shaded boxes correspond to elements that belong to the indexing structure, while the transparent boxes correspond to the accessible entries in the MIB.

The DOORS MIB defines three tables. The first table (*rosClientTable*)[1] lists the client objects that make use of DOORS services, and some of its parameters, like name, replication style, number of replications. As an example, in Figure 2 *rosClientName* corresponds to O1 and O2, where the value of *rosClientReplDegree* for O1 is 2.

The second table (*rosObjectTable*) lists the objects and replicas under control of the ReplicaManager for each client, and contains entries for the host name, object status, etc. In Figure 2, the value for *rosObjectHost* for the one instance of O2 is M1.

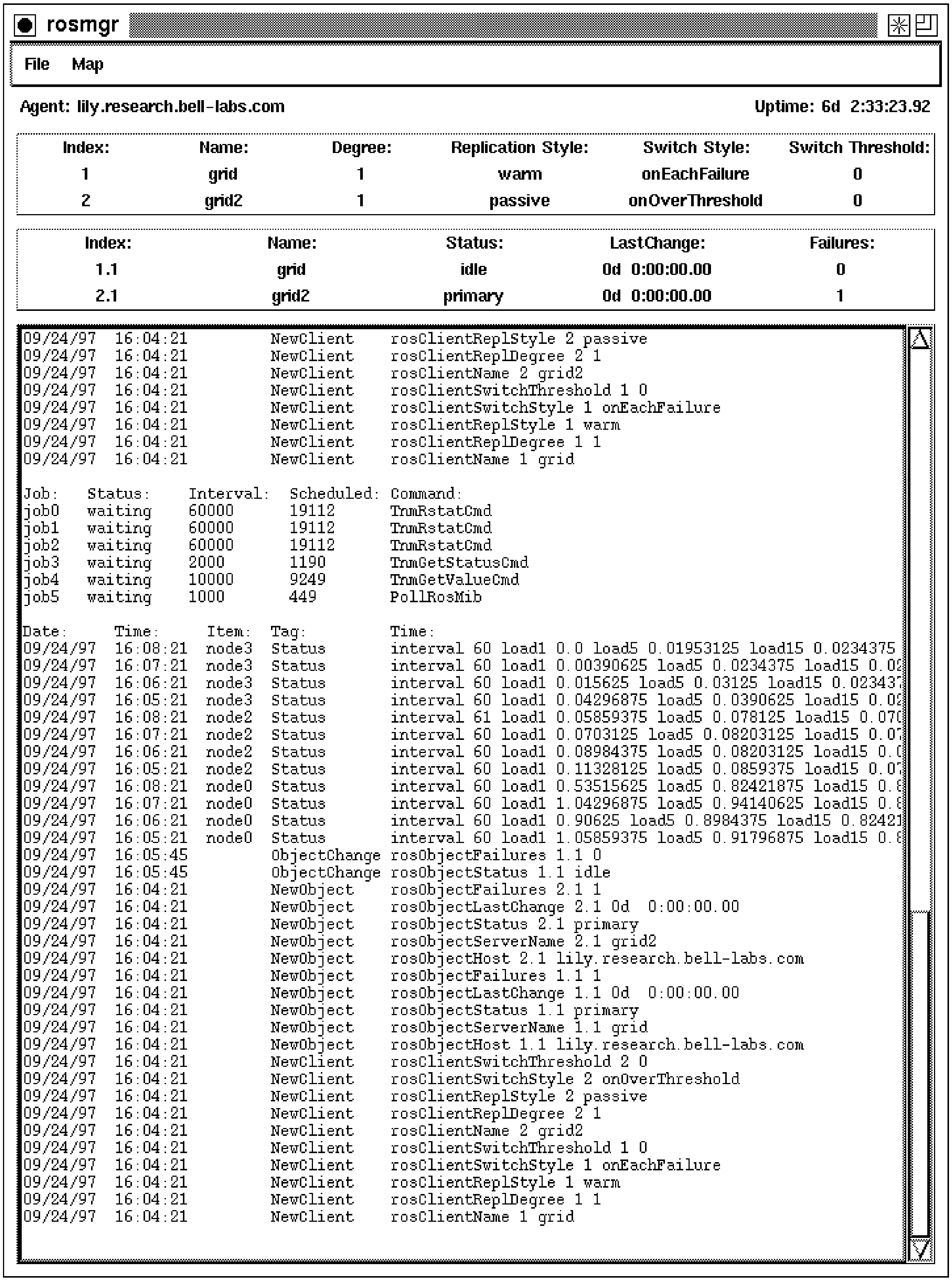
The third table (*rosCompTable*) lists all the components that make up DOORS, together with their location. For instance, the entry *rosCompType* can take values 1 to 3 depending on whether a component is a WatchDog, a SuperWatchDog, or the ReplicaManager. This table is useful for the manager to discover the structure of DOORS, so that it can adjust the monitoring activities to the most critical components.

C Manager

A manager has been implemented which displays the DOORS MIB data, as well as other system data collected, in a graphical user interface (Figure 4). The implementation uses the Tcl/Tk extension provided by the Tnm management package [13]. The Tnm Tcl extension allows fast development of a management front-end, including a graphical interface. Figure 4 shows DOORS running for two CORBA clients, “grid” and “grid2”. The top frame shows data collected about the clients, such as their name, and replication and switch style. The frame below that shows information about the object instances. In this case both clients have a single instance running, and it can be seen that “grid2” experienced one failure. Finally, the large frame in the manager interface continuously displays data that is being collected.

The DoorMan manager uses a completely event driven approach, where low-level monitoring functions generate internal events that are passed along the network topology and activate event bindings. These bindings allow the man-

[1]Note that the MIB uses the earlier acronym ROS (Reliable Object Service) instead of DOORS.



Agent: lily.research.bell-labs.com Uptime: 6d 2:33:23.92

Index:	Name:	Degree:	Replication Style:	Switch Style:	Switch Threshold:
1	grid	1	warm	onEachFailure	0
2	grid2	1	passive	onOverThreshold	0

Index:	Name:	Status:	LastChange:	Failures:
1.1	grid	idle	0d 0:00:00.00	0
2.1	grid2	primary	0d 0:00:00.00	1

```
09/24/97  16:04:21          NewClient    rosClientReplStyle 2 passive
09/24/97  16:04:21          NewClient    rosClientReplDegree 2 1
09/24/97  16:04:21          NewClient    rosClientName 2 grid2
09/24/97  16:04:21          NewClient    rosClientSwitchThreshold 1 0
09/24/97  16:04:21          NewClient    rosClientSwitchStyle 1 onEachFailure
09/24/97  16:04:21          NewClient    rosClientReplStyle 1 warm
09/24/97  16:04:21          NewClient    rosClientReplDegree 1 1
09/24/97  16:04:21          NewClient    rosClientName 1 grid

Job:   Status:    Interval:  Scheduled: Command:
job0   waiting    60000      19112      TnmRstatCmd
job1   waiting    60000      19112      TnmRstatCmd
job2   waiting    60000      19112      TnmRstatCmd
job3   waiting    2000       1190       TnmGetStatusCmd
job4   waiting    10000      9249       TnmGetValueCmd
job5   waiting    1000       449        PollRosMib

Date:     Time:     Item:  Tag:         Time:
09/24/97  16:08:21  node3  Status       interval 60 load1 0.0 load5 0.01953125 load15 0.0234375
09/24/97  16:07:21  node3  Status       interval 60 load1 0.00390625 load5 0.0234375 load15 0.02
09/24/97  16:06:21  node3  Status       interval 60 load1 0.015625 load5 0.03125 load15 0.02343
09/24/97  16:05:21  node3  Status       interval 60 load1 0.04296875 load5 0.0390625 load15 0.02
09/24/97  16:08:21  node2  Status       interval 61 load1 0.05859375 load5 0.078125 load15 0.070
09/24/97  16:07:21  node2  Status       interval 60 load1 0.0703125 load5 0.08203125 load15 0.07
09/24/97  16:06:21  node2  Status       interval 60 load1 0.08984375 load5 0.08203125 load15 0.0
09/24/97  16:05:21  node2  Status       interval 60 load1 0.11328125 load5 0.0859375 load15 0.07
09/24/97  16:08:21  node0  Status       interval 60 load1 0.53515625 load5 0.82421875 load15 0.8
09/24/97  16:07:21  node0  Status       interval 60 load1 1.04296875 load5 0.94140625 load15 0.8
09/24/97  16:06:21  node0  Status       interval 60 load1 0.90625 load5 0.8984375 load15 0.82421
09/24/97  16:05:21  node0  Status       interval 60 load1 1.05859375 load5 0.91796875 load15 0.8
09/24/97  16:05:45         ObjectChange rosObjectFailures 1.1 0
09/24/97  16:05:45         ObjectChange rosObjectStatus 1.1 idle
09/24/97  16:04:21         NewObject    rosObjectFailures 2.1 1
09/24/97  16:04:21         NewObject    rosObjectLastChange 2.1 0d  0:00:00.00
09/24/97  16:04:21         NewObject    rosObjectStatus 2.1 primary
09/24/97  16:04:21         NewObject    rosObjectServerName 2.1 grid2
09/24/97  16:04:21         NewObject    rosObjectHost 2.1 lily.research.bell-labs.com
09/24/97  16:04:21         NewObject    rosObjectFailures 1.1 1
09/24/97  16:04:21         NewObject    rosObjectLastChange 1.1 0d  0:00:00.00
09/24/97  16:04:21         NewObject    rosObjectStatus 1.1 primary
09/24/97  16:04:21         NewObject    rosObjectServerName 1.1 grid
09/24/97  16:04:21         NewObject    rosObjectHost 1.1 lily.research.bell-labs.com
09/24/97  16:04:21         NewClient    rosClientSwitchThreshold 2 0
09/24/97  16:04:21         NewClient    rosClientSwitchStyle 2 onOverThreshold
09/24/97  16:04:21         NewClient    rosClientReplStyle 2 passive
09/24/97  16:04:21         NewClient    rosClientReplDegree 2 1
09/24/97  16:04:21         NewClient    rosClientName 2 grid2
09/24/97  16:04:21         NewClient    rosClientSwitchThreshold 1 0
09/24/97  16:04:21         NewClient    rosClientSwitchStyle 1 onEachFailure
09/24/97  16:04:21         NewClient    rosClientReplStyle 1 warm
09/24/97  16:04:21         NewClient    rosClientReplDegree 1 1
09/24/97  16:04:21         NewClient    rosClientName 1 grid
```

Figure 4: Screen dump of DoorMan manager application.

ager to analyze the event in the current context and generate other events if some conditions are met. Hence, higher level management information can be computed by passing events through a set of event bindings.

V Conclusions and Topics for Further Research

This paper discusses the design and implementation of DoorMan, a management interface for DOORS. DoorMan allows for monitoring the operation of DOORS, as well as for feeding back to DOORS information abstracted from the underlying computing platform. DoorMan is based on SNMP, which allows us to use off-the-shelf agent and management application software.

Most critical in the design of DoorMan is the interface between SNMP and CORBA/ DOORS. We designed a subagent that can bind with the DOORS ReplicaManager, and we specified a MIB for DOORS. The ReplicaManager's IDL has been extended to allow the sub-agent to collect data, and to allow it to pass information back.

The current implementation of DoorMan allows future experimenting with different management schemes. Different approaches are possible when determining which data to collect about the underlying computing platform. To determine this, we can make use of insights we obtained from the statistical analysis of the correlation between measured objects and system failures [3]. Furthermore, it is of interest to determine how to extract information about the underlying computing system at the desired level of abstraction, and pass it back to DOORS. Different algorithms can then be developed that use this information. The extraction of information as well as the algorithms deal with the trade-off between simplicity and quality of decision making, which needs further experimental research to be fully understood.

REFERENCES

[1] P. E. Chung, Y. Huang, S. Yajnik, D. Liang, and J. Shih, "DOORS: providing fault tolerance for CORBA applications," Submitted for publication, December 1997.

[2] M. Daniele, B. Wijnen, and D. Francisco, "Agent extensibility (AgentX) protocol version 1," Internet Draft <draft-ietf-agentx-ext-pro-04.txt>, Digital Equipment Corporation, IBM, Cisco Systems, June 1997.

[3] S. Garg, A. P. A. van Moorsel, K. S. Trivedi, and K. Vaidyanathan, "Age estimation and failure forecasting in software systems using time-series analysis," Submitted for publication, December 1997.

[4] IONA Technologies Ltd., "Orbix reference guide," Technical Report Release 2.1, October 1995.

[5] S. Maffeis, "Piranha - a CORBA tool for high availability," *IEEE Computer*, April 1997.

[6] S. Maffeis and D. C. Schmidt, "Constructing reliable distributed communication systems with CORBA," *IEEE Communications Magazine*, vol. 14, no. 2, , February 1997.

[7] S. Mazumdar, "Inter-domain management between CORBA and SNMP: Web-based management–CORBA/SNMP gateway approach," in *Seventh International Workshop on Distributed Systems: Operations & Management.* IFIP/IEEE, October 1996.

[8] OMG, "The Common Object Request Broker: Architecture and specification," Technical Report Revision 2.0, Object Management Group, 1995.

[9] OMG, "CORBAServices: Common object services specification," Technical report, Object Management Group, March 1995.

[10] Open Group, "Inter-domain management specifications: Specification translation," Preliminary Specification P509, Open Group, March 1997.

[11] D. Perkins and E. McGinnis, *Understanding SNMP MIBs*, Prentice-Hall, Upper Saddle River, NJ, USA, 1997.

[12] M. T. Rose, *The Simple Book: An Introduction to Networking Management*, Series in Innovative Technology, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1996.

[13] J. Schönwälder and H. Langendörfer, "Tcl extensions for network management applications," in *Proc. 3rd Tcl/Tk Workshop*, pp. 279–288, Toronto (Canada), July 1995.

[14] W. Stallings, *SNMP, SNMPv2, and CMIP, The Practical Guide to Network-Management Standards*, Addison-Wesley, Reading, MA, USA, 1993.